routovanie

cieľ: doručovať srávy medzi ľubovoľnou dvojicou

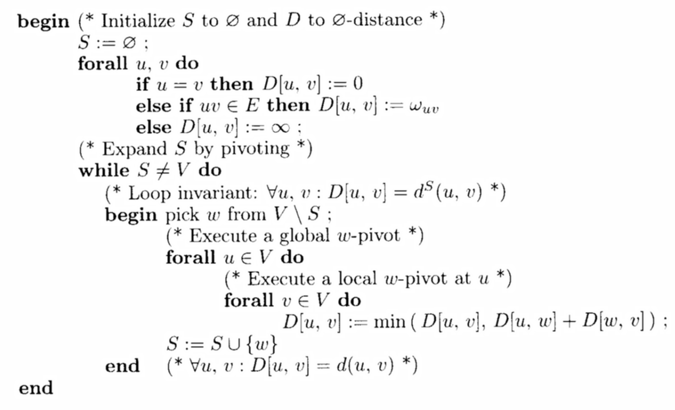
modely

- destination based
- splittable
- connections (wormhole)
- buffering
- selfish

ciele

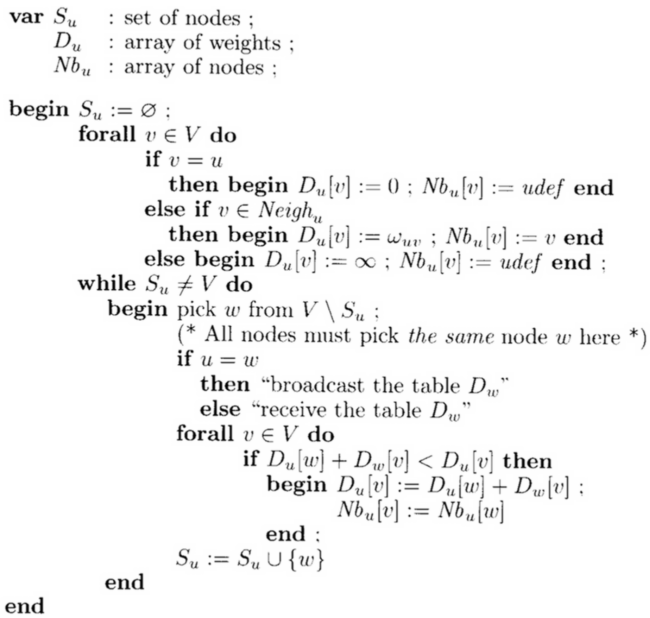
- statické váhy (najkratšie cesty)
- dynamické váhy (hot potato)
- deadlock

najkratšie cesty



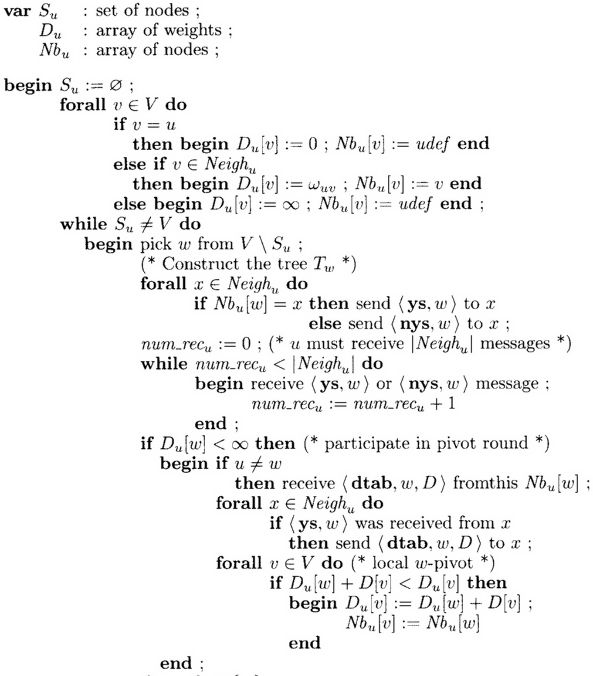
```
begin (* Initialize $S$ to $\varnothing$ and $D$ to $\varnothing$-distance *)
        $S := \varnothing$ ;
        forall $u, v$ do
                if $u = v$ then $D[u, v] := 0$
                else if $uv \in E$ then $D[u, v] := \omega_{uv}$
                else $D[u, v] := \infty$ ;
        (* Expand $S$ by pivoting *)
        while $S \neq V$ do
            (* Loop invariant: $\forall u, v : D[u, v] = d^S(u, v)$ *)
            begin pick $w$ from $V \setminus S$ ;
                    (* Execute a global $w$-pivot *)
                    forall $u \in V$ do
                            (* Execute a local $w$-pivot at $u$ *)
                            forall $v \in V$ do
                                    $D[u, v] := \min(D[u, v], D[u, w] + D[w, v])$ ;
                    $S := S \cup \{w\}$
            end     (* $\forall u, v : D[u, v] = d(u, v)$ *)
end
```

najkratšie cesty



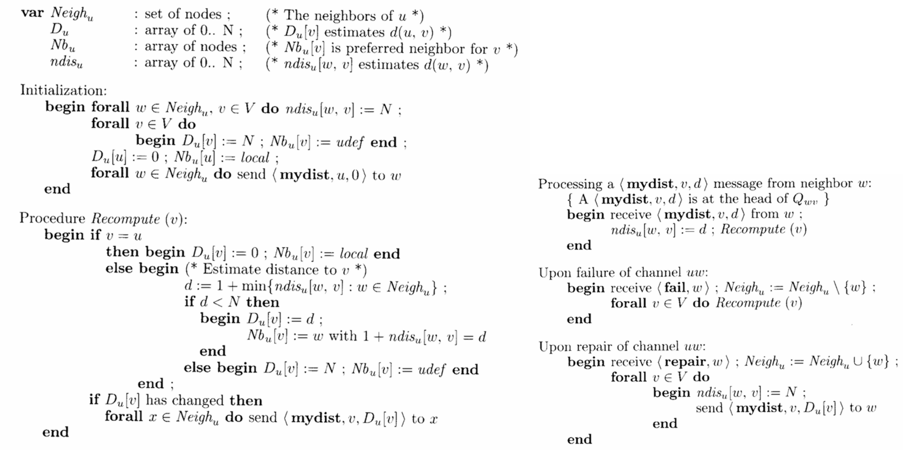
```
var $S_u$    : set of nodes ;
    $D_u$    : array of weights ;
    $Nb_u$   : array of nodes ;

begin $S_u := \varnothing$ ;
      forall $v \in V$ do
             if $v = u$
                then begin $D_u[v] := 0$ ; $Nb_u[v] := udef$ end
             else if $v \in Neigh_u$
                then begin $D_u[v] := \omega_{uv}$ ; $Nb_u[v] := v$ end
             else begin $D_u[v] := \infty$ ; $Nb_u[v] := udef$ end ;
      while $S_u \neq V$ do
         begin pick $w$ from $V \setminus S_u$ ;
               (* All nodes must pick the same node $w$ here *)
               if $u = w$
                  then "broadcast the table $D_w$"
                  else "receive the table $D_w$"
               forall $v \in V$ do
                      if $D_u[w] + D_w[v] < D_u[v]$ then
                         begin $D_u[v] := D_u[w] + D_w[v]$ ;
                               $Nb_u[v] := Nb_u[w]$
                         end ;
               $S_u := S_u \cup \{w\}$
         end
end
```

najkratšie cesty



```
var $S_u$    : set of nodes ;
    $D_u$    : array of weights ;
    $Nb_u$   : array of nodes ;

begin $S_u := \varnothing$ ;
      forall $v \in V$ do
            if $v = u$
               then begin $D_u[v] := 0$ ; $Nb_u[v] := udef$ end
            else if $v \in Neigh_u$
               then begin $D_u[v] := \omega_{uv}$ ; $Nb_u[v] := v$ end
            else begin $D_u[v] := \infty$ ; $Nb_u[v] := udef$ end ;
      while $S_u \neq V$ do
         begin pick $w$ from $V \setminus S_u$ ;
               (* Construct the tree $T_w$ *)
               forall $x \in Neigh_u$ do
                     if $Nb_u[w] = x$ then send ⟨ ys, $w$ ⟩ to $x$
                                      else send ⟨ nys, $w$ ⟩ to $x$ ;
               $num\_rec_u := 0$ ; (* $u$ must receive $|Neigh_u|$ messages *)
               while $num\_rec_u < |Neigh_u|$ do
                     begin receive ⟨ ys, $w$ ⟩ or ⟨ nys, $w$ ⟩ message ;
                           $num\_rec_u := num\_rec_u + 1$
                     end ;
               if $D_u[w] < \infty$ then (* participate in pivot round *)
                  begin if $u \neq w$
                           then receive ⟨ dtab, $w, D$ ⟩ fromthis $Nb_u[w]$ ;
                        forall $x \in Neigh_u$ do
                              if ⟨ ys, $w$ ⟩ was received from $x$
                                 then send ⟨ dtab, $w, D$ ⟩ to $x$ ;
                        forall $v \in V$ do (* local $w$-pivot *)
                              if $D_u[w] + D[v] < D_u[v]$ then
                                 begin $D_u[v] := D_u[w] + D[v]$ ;
                                       $Nb_u[v] := Nb_u[w]$
                                 end
                  end ;
```

Netchange



```
var $Neigh_u$ : set of nodes ;     (* The neighbors of $u$ *)
    $D_u$     : array of 0.. N ;   (* $D_u[v]$ estimates $d(u, v)$ *)
    $Nb_u$    : array of nodes ;   (* $Nb_u[v]$ is preferred neighbor for $v$ *)
    $ndis_u$  : array of 0.. N ;   (* $ndis_u[w, v]$ estimates $d(w, v)$ *)

Initialization:
    begin forall $w \in Neigh_u$, $v \in V$ do $ndis_u[w, v] := N$ ;
          forall $v \in V$ do
                begin $D_u[v] := N$ ; $Nb_u[v] := udef$ end ;
          $D_u[u] := 0$ ; $Nb_u[u] := local$ ;
          forall $w \in Neigh_u$ do send ⟨ mydist, $u, 0$ ⟩ to $w$
    end

Procedure Recompute ($v$):
    begin if $v = u$
              then begin $D_u[v] := 0$ ; $Nb_u[v] := local$ end
              else begin (* Estimate distance to $v$ *)
                        $d := 1 + \min\{ndis_u[w, v] : w \in Neigh_u\}$ ;
                        if $d < N$ then
                           begin $D_u[v] := d$ ;
                                 $Nb_u[v] := w$ with $1 + ndis_u[w, v] = d$
                           end
                        else begin $D_u[v] := N$ ; $Nb_u[v] := udef$ end
                   end ;
          if $D_u[v]$ has changed then
             forall $x \in Neigh_u$ do send ⟨ mydist, $v, D_u[v]$ ⟩ to $x$
    end
```

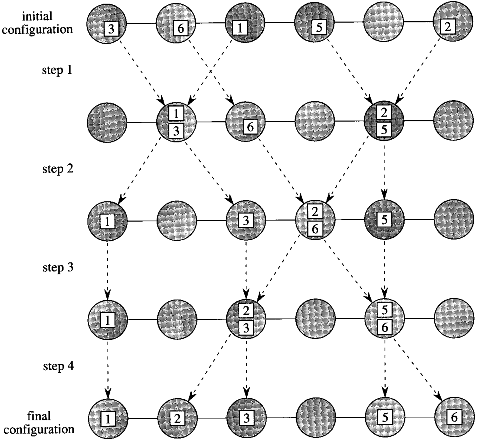
```
Processing a ⟨ mydist, $v, d$ ⟩ message from neighbor $w$:
    { A ⟨ mydist, $v, d$ ⟩ is at the head of $Q_{wu}$ }
    begin receive ⟨ mydist, $v, d$ ⟩ from $w$ ;
          $ndis_u[w, v] := d$ ; Recompute ($v$)
    end

Upon failure of channel $uw$:
    begin receive ⟨ fail, $w$ ⟩ ; $Neigh_u := Neigh_u \setminus \{w\}$ ;
          forall $v \in V$ do Recompute ($v$)
    end

Upon repair of channel $uw$:
    begin receive ⟨ repair, $w$ ⟩ ; $Neigh_u := Neigh_u \cup \{w\}$ ;
          forall $v \in V$ do
                begin $ndis_u[w, v] := N$ ;
                      send ⟨ mydist, $v, D_u[v]$ ⟩ to $w$
                end
    end
```

korektnosť

lexikograficky klesá hodnota $[t_0, t_1, \ldots, t_N]$
kde t_i je počet správ $\langle$**mydist**$, i\rangle +$ počet dvojíc u, v kde $D_u[v] = i$

packet routing

- synchrónny režim
- vrcholy majú pakety (uložené v bufferoch)
- v jednom kroku po jednej linke ide max. jeden paket
- algoritmus = odchádzajúce linky + priorita bufferov
- celkový čas



packet routing na mriežke $\sqrt{N} \times \sqrt{N}$

vstup

Každý vrchol má 1 paket, do každého smeruje 1 paket (permutation routing)

algoritmus

Najprv riadok, potom stĺpec. Prednosť má ten s najdlhšou cestou.

analýza: stačí $2\sqrt{N} - 2$ krokov

- po $\sqrt{N} - 1$ krokoch je každý v správnom stĺpci (nebrzdia sa)
- routovanie v stĺpci ide v $\sqrt{N} - 1$ krokoch
  - pre každé i platí: po $N - 1$ krokoch sú koncové pakety na koncových miestach
  - dôvod: zdržujú sa iba navzájom

veľkosť buffra v najhoršom prípade: $2/3\sqrt{N} - 3$

veľkosť buffra: priemerný prípad I

setting

Každý vrchol má jeden paket s **náhodným cieľom**

max. veľkosť buffra $\approx$ počet zahnutí vo vrchole

psť, že aspoň r zahne $\leq \binom{\sqrt{N}}{r}\left(\frac{1}{\sqrt{N}}\right)^r < \left(\frac{e}{r}\right)^r$

pre $r = \frac{e \log N}{\log \log N}$ je psť $o(N^{-2})$